TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MIIKA HEINONEN

MOBIILITULOSTUKSEN TOTEUTTAMINEN IOS-LAITTEILLE SWIFT-OHJELMOINTIKIELELLÄ

Kandidaatintyö

Tarkastaja: Sakari Lahti
Jätetty tarkastettavaksi 10.1.2017

TIIVISTELMÄ

Miika Heinonen: Mobiilitulostuksen toteuttaminen iOS-laitteille Swift-ohjelmointikielellä
Tampereen teknillinen yliopisto
Kandidaatintyö, 17 sivua
Tammikuu 2017
Tietotekniikan koulutusohjelma
Pääaine: Ohjelmistotekniikka
Tarkastaja: Sakari Lahti

Avainsanat: Mobiilisovellus, iOS, Swift, Objective-C, mobiilitulostus, kotinäytteenotto

Työssä kuvataan kotinäytteenoton mobiilisovelluksen mobiilitulostusominaisuuden suunnittelu ja toteutus. Lisäksi verrataan työn pohjana toimivan mobiilisovelluksen toteutustapaa muihin mahdollisiin toteutustapoihin. Työssä kuvattu ohjelma on hybridiohjelma, eli se koostuu selainmoottorista joka pyörittää web-ohjelmaa ja natiiviohjelmasta, joka vastaa ohjelman laitteistoa vaativista toimenpiteistä ja pyörittää myös selainmoottoria. Muita mahdollisia vaihtoehtoja olisi ollut natiivisovellus tai web-sovellus. Sovellus oltaisiin voitu kehittää myös monialustasovelluskehyksellä, joten sitä verrataan myös monialustatoteutukseen. Työssä myös esitellään kehityskielenä käytetyn Swfitin perusteet ja perusperiaatteet.

Kotinäytteenotossa on kyse verinäytteen ottamisesta potilaan luona, laboratorion ulkopuolella. Työn liikkuvan luonteen takia näytteitä ottaville hoitajille on ollut tarpeen kehittää mahdollisimman helppokäyttöinen ja helposti liikuteltava laitteisto.

Työssä selvisi, että ohjelman toteutustapa oli varsinkin alkutilanteen huomioon ottaen varsin onnistunut, sillä käytettävissä oli aikaisemman kotinäytteenotto-ohjelmisto, joka on web-ohjelma. Web-ohjelma pyörittäminen natiiviohjelman selainmoottorissa on tehokkuudeltaan riittävä ja toteutuksena helpoin mahdollinen. Monialustatoteutus olisi varteenotettavin vaihtoehto, sillä sen kehityksessä olisi myös mahdollista käyttää jo olemassa olevaa web-ohjelmaa ja samalla saavuttaa lisähyötyä, jos sovellus on myöhemmin tarkoitus kääntää myös muille alustoille. Ohjelmaan toteutettu mobiilitulostusominaisuus todettiin toimivaksi ja helppokäyttöiseksi.

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

CSS	Cascading Style Sheets, verkkosivun tyyliohjeiden laji
HTML	Hypertext markup language, verkkosivun merkintäkieli
iOS	Applen kehittämä käyttöjärjestelmä
MIT	Massachusetts Institute of Technology
MVC	Model-View-Controller-arkkitehtuuri, ohjelmostoarkkitehtuuri
SDK	engl. Software Development Kit, Sovelluskehitystyökalut
URL	engl. Uniform Resource Locator, verkkosivun osoite

1. JOHDANTO

Näytteenottaminen tarkoittaa verinäytteen ottamista potilaasta [1]. Yleensä verinäytteet otetaan sairaalassa tai terveyskeskuksessa erillisissä laboratoriotiloissa. Kotinäytteenotolla tarkoitetaan näytteen ottamista, joka tapahtuu potilaan luona [1]. Toistaiseksi kotinäytteenottoa tarjotaan vain valituille potilaille, jotka on todettu liian huonokuntoisiksi laboratoriotiloissa tapahtuvalle näytteenotolle. Jatkossa kotinäytteenottoa on tarkoitus tarjota myös kaikille halukkaille, jolloin potilaiden ei tarvitsisi lähteä sairastuttuaan terveyskeskukseen. Koska näytteenotto tapahtuu laboratorion ulkopuolella ja koska työ on luonteeltaan liikkuvaa, on käytettävien laitteiden helppokäyttöisyydellä ja helpolla kuljettamisella iso merkitys. Kotinäytteenoton mobiiliohjelma, johon tässä työssä lisättiin Bluetooth-tulostus, vastaa tähän tarpeeseen antamalla hoitajalle mahdollisuuden kirjata otetut näytteet helposti mobiililaitteella.

Tämän työn tarkoituksena on kuvata kotinäytteenoton mobiilisovelluksen Bluetooth-tulostuksen toteutus iOS-käyttöjärjestelmälle käyttäen Applen uutta Swift-ohjelmointikieltä. Nykyisessä järjestelmässä näytteen ottanut hoitaja ottaa valmiin viivakooditarran, skannaa sen mobiililaitteella ja kirjaa näytteen tiedot järjestelmään. Työn kuvaamassa projektissa järjestelmää muutetaan siten, että viivakoodit tulostetaan kirjaamisen yhteydessä, jolloin säästetään aikaa, sillä viivakoodin lukeminen on toistaiseksi tapahtunut mobiililaitteen kameralla. Tämä on suhteellisen hidasta verrattuna viivakoodin lukemiseen erillisellä viivakoodilukijalla.

Työn kuvaamassa projektissa on tarkoitus lisätä järjestelmään mahdollisuus tulostaa erillisellä mobiilitulostimella tarra juuri otetun näytteen näyteputken kylkeen. Työssä on tarkoitus arvioida Swiftin, ja sillä toteutettavien natiivisovellusten, soveltuvuutta vastaaviin projekteihin myös jatkossa. Kilpailevia ratkaisuja on kuitenkin jo useita, aina muilla tekniikoilla toteutetuista natiivisovelluksista erilaisiin web-toteutuksiin [2]. Työssä toteuttavan ohjelman tuli olla riittävän nopea ja varmatoiminen, kun taas kehitysympäristön ja kielen tuli olla riittävän helppokäyttöisiä ja selkeitä. Toteutusta ja valittua toteutustapaa verrattiin myös muihin mahdollisiin ratkaisuihin. Vaikka mobiilisovellus käsittää koko näytteenottoprosessin, keskitytään tässä työssä nimenomaan mobiilitulostuksen toteutukseen. Arvioitaessa toteutustapaa arvioidaan kuitenkin koko sovellusta.

Työssä on kuusi lukua. Toisessa luvussa kuvataan kotinäytteenottoon liittyvä toimintaympäristö, eli minkälaisessa ympäristössä kehitettyä ohjelmaa käytetään. Toimintaympäristöön kuuluu tärkeänä osana myös käytettävät laitteet. Kolmannessa luvussa avataan työssä käytettyä Swift-ohjelmointikieltä ja vertaillaan sitä edeltäjäänsä Objective-C:hen,

jolla työssä käytetyt tulostinvalmistajan kirjastot on toteutettu. Neljännessä luvussa kuvataan ohjelman varsinainen toteutus, rakenne ja tehdyt suunnitteluratkaisut. Viidennessä luvussa esitellään muut toteutustapavaihtoehdot ja verrataan niitä toteutettuun sovellukseen. Kuudennessa luvussa tehdään yhteenveto toteutetun ohjelman toiminnasta ja vertaillaan sovelluksen toteutustapaa vaihtoehtoisiin toteutustapoihin.

2. KOTINÄYTTEENOTON TOIMINTAYMPÄRISTÖ

Kotinäytteenoton tapahtuminen laboratorion ulkopuolella tuo mukanaan omat haasteensa. Koska näytteitä ottavat hoitajat kiertävät potilaiden luona ottamassa näytteet, tulee tarvittavien välineiden olla helposti mukana kulkevia. Tämä koskee myös näytteiden kirjaamiseen käytettävää laitteistoa. Tästä syystä hoitajia varten on kehitetty, toistaiseksi kehitysvaiheessa oleva, mobiilisovellus, josta voidaan lukea mitä näytteitä pitää päivän aikana ottaa ja lopulta kirjata otetut näytteet. Mobiilisovelluksen pohjana toimii aikaisempi kotinäytteenottosovellus, joka on toteutettu web-ohjelmana. Ohjelma vaatii luonnollisesti pääsyn tietokantoihin ja nykyisessä versiossaan itse sovelluskin ajetaan verkon yli. Tästä syystä käytettävän mobiililaitteen tulee toimia vähintään langattomassa lähiverkossa, tai mieluiten mobiiliverkon datayhteydellä.

2.1 Kotinäytteenotto

Kotinäytteitä ottavat hoitajat kiertävät päivän aikana useiden eri potilaiden luona. Aamulla laboratoriolta lähtiessään hoitajat hakevat reittisuunnitelman ja tarvitsemansa välineet. Potilaan luokse saavuttuaan hoitaja tarkistaa potilaan henkilöllisyyden, valmistelee potilaan näytteenottoa varten ja ottaa näytteen [1]. Otettuja näytteitä palautetaan kierron aikana laboratoriolle sopivissa väleissä. Otetut verinäytteet yksilöidään viivakooditarroilla. Työn kuvaamaa uudistusta edeltävässä versiossa hoitaja on tulostanut viivakooditarrat jo kierrolle lähtiessään. Kiertoa valmistellessaan hoitajan on myös täytynyt huolehtia, että oikeat viivakooditarrat liitetään niitä vastaaviin näytepyyntötulosteisiin. Yhtenä vaihtoehtona viivakooditarroille on tarjottu myös valmiiksi viivakooditettuja putkia, joka olisi työssä kuvatulle sovellukselle korvaava ratkaisu [1]. Tämän työn mukaisen mobiilitulostusta käyttävän ohjelman etuna olisi ainakin se, että näytteenottoaika saataisiin kirjattua tarkasti ja automaattisesti.

2.2 Käytetyt laitteet

Kotinäytteenoton mobiilisovellus tarvitsee toimiakseen mobiililaitteen, jossa sovellusta ajetaan ja mobiilitulostimen, jolla tulostetaan näyteputkiin kiinnitettävät näytetarrat. Mobiililaitteessa ajettava sovellus vastaa näytteiden ja potilaan tietojen näyttämisestä ja tulosteiden valmistelusta tulostinta varten. Mobiililaitteelle olisi ollut luonnollisesti useita vaihtoehtoja, mutta koska aiempi kehitysversio on kehitetty Applen iOS-laitteita varten, on uudistuksetkin järkevää tehdä samalle alustalle osaksi edellistä järjestelmää, kokonaan uuden alustan käyttöönottamisen sijaan. Tulostinvalintaan vaikuttavia tekijöitä ovat sen liitettävyys ja kannettavuus. Sinänsä sopivia yhteystapoja mobiililaitteen ja tulostimen välillä on useita, kuten langaton lähiverkko, Bluetooth ja kaapeli. Langaton lähiverkko on kotinäytteenottosovelluksen tapauksessa rajattava pois, sillä ainakin työssä lopulta

käytettyjen Datamax-tulostinten tapauksessa tulostin itsessään toimii langattoman verkon tukiasemana, johon mobiililaitteen tulisi yhdistää. Laitteet, joilla sovellusta aiotaan käyttää eivät kuitenkaan tue useaa samanaikaista langatonta lähiverkkoyhteyttä, joten ne voisivat olla yhteydessä joko tulostimeen tai internetiin, eivät molempiin yhtä aikaa. Toimiva internetyhteys on sovelluksen toiminnalle kuitenkin erittäin tärkeää, joten langaton lähiverkko ei tule kysymykseen. Kaapeleita käyttävä yhteystapa olisi toimiva, mutta sitä tukevia tulostimia oli sovellusta kehitettäessä tarjolla rajoitetusti, jos ollenkaan. Yhteystavaksi valikoitui lopulta Bluetooth, sen helppokäyttöisyyden ja sitä tukevien tulostimien saatavuuden vuoksi.

Tulostimena työssä käytetään Datamax Microflash 4te-mobiilitulostinta, joka on kannettava tulostin, jolla voidaan tulostaa viivakooditarroja. Tulostin tukee tulostamista sekä langattoman lähiverkon, että Bluetooth-yhteyden avulla. Työssä käytetään Bluetooth-tulostusta. Tulostin ei tunnistu suoraan tulostinlaitteena, vaan tulostimelle annetaan komentoja erillisen rajapinnan kautta. Datamaxin tulostimilleen tarjoamat kehitystyökalut ovat melko rajoittuneet, sillä kehittäjälle annetaan lähinnä kirjasto, jossa on tarvittavat luokat tulostimen ohjaamiseen [3]. Kirjasto itsessään on dokumentoitu riittävän tarkasti, mutta tämän lisäksi on tarjolla vain yksinkertaiset esimerkit tärkeimmistä toiminnoista, kuten yhteydenotosta ja yksinkertaisesta testitulosteesta.

Ohjelmaa voidaan käyttää useilla erilaisilla iOS-laitteilla. Applen iOS-käyttöjärjestelmä on valikoitunut pääasiassa tuttuuden vuoksi, sillä vastaavissa tehtävissä on aiemmin käytetty Applen iPod-laitteita, joissa käytetään samaa käyttöjärjestelmää. iPodien tuotanto on kuitenkin nykyisellään lopetettu, joten iPhone on luonnollinen seuraaja, vaikka toistaiseksi laitteen puhelinominaisuuksille ei käyttöä olekaan. Kehityksen aikana testilaitteena käytettin Apple iPhone SE-matkapuhelinta. Kyseinen matkapuhelin on 4-tuumainen ja siinä on Apple A9-mikropiiri ja 12 megapikselin kamera, jota käytetään viivakoodien skannaamiseen. Työn kannalta tärkeitä ovat laitteen Bluetooth-ominaisuudet. Matkapuhelin tukisi uudempaa Bluetooth Low Energy-tekniikkaa, mutta koska käytetyt tulostimet eivät sitä tue, käytetään työssä perinteistä Bluetooth-standardia. Matkapuhelin tukee myös useita mobiilidatatekniikoita, joten ohjelman toiminannan kannalta välttämätön internet-yhteys on helppo muodostaa. Puhelimessa on iOS 10 – käyttöjärjestelmä, mutta ohjelma on mahdollista kääntää usealle muullekin käyttöjärjestelmäversiolle.

3. OHJELMOINTIKIELET JA KEHITYSYMPÄRISTÖ

Työssä on pääasiallisena kehityskielenä käytetty Applen Swift ohjelmointikieltä, joten kehitysympäristöksi on luonnollisesti valikoitunut Applen Xcode. Swiftin lisäksi työssä on oltu tekemisissä Applen toisen iOS-laitteille suunnatun ohjelmointikielen, Objective-C:n kanssa, sillä tulostimen kirjastot on toteutettu sillä. Swiftin ollessa vielä verrattain uusi kieli, tulostimen kehitystyökalut tai niiden dokumentaatio ei tarjoa esimerkkejä tai muutakaan materiaalia Swift-kehitystä varten.

3.1 Swift-ohjelmointikieli

Swift on moniparadigmainen käännettävä ohjelmointikieli, jonka on kehittänyt Apple inc. vuonna 2014. Swift on uusin Applen laitteiden ohjelmistokehitykseen tarkoitettu kieli, jonka on tarkoitus jatkaa edeltäjänsä Objective-C:n rinnalla [4]. Swiftistä on pyritty tekemään Objective-C:tä turvallisempi ja yksinkertaisempi. Tässä työssä käytetään Swiftin kolmatta versiota.

Swift sisältää myös paljon kokonaan uusia ominaisuuksia, joilla on pyritty lisäämään kielen käytettävyyttä nykyaikaisessa ohjelmistokehityksessä. Playground-tila mahdollistaa ohjelman testaamisen reaaliajassa. Ohjelmakoodi siis ajetaan välittömästi, kun siihen on tehty muutoksia. Tämä mahdollistaa yksittäisten algoritmien nopean ja tehokkaan testaamisen, koko ohjelman kääntämisen sijaan. Perinteisen operaattorien lisäksi Swiftissä on olemassa väli-operaattorit, ylivuoto-operaattorit ja mahdollisuus määrittää omia operaattoreita olemassa olevien aritmeettisten merkkien avulla. [5]

3.1.1 Swift ja Objective-C

Objective-C on Swiftin edeltäjä, jonka ensimmäinen versio on kehitetty jo vuonna 1984. Alun perin sitä käytettiin Applen OS X:n edeltäjän, NeXTSTEP-käyttöjärjestelmän kehityksessä. Koska Objective-C:llä on mahdollista toteuttaa samanlaisia iOS-sovelluksia, kuin Swiftillä, on mielekästä tutustua näiden kahden eroihin.

Objective-C ja Swift käyttävät samaa kääntäjää (The Low Level Virtual Machine, LLVM), joka kääntää Swift-lähdekoodin valitulle laitteelle optimoiduksi lähdekoodiksi. Swift on täysin yhteensopiva Objective-C:n kanssa, sillä sen kanssa voi käyttää kaikkia Objective-C:n kirjastoja ja sen tukemia ominaisuuksia [5]. Objective-C:n koodia on myös mahdollista käyttää Swiftissä sellaisenaan. Samoin Swift-koodia on mahdollista viedä Objective-C-projektiin. Edellä mainitut ominaisuudet ovat todella hyödyllisiä varsinkin Swiftin ollessa vielä verrattain tuore ohjelmointikieli, jolle ei vielä ole toteutettu omia versioita Objective-C:n vastaavista. Tässä työssä käytettiin tulostinvalmistajan Objective-C:llä toteutettuja kirjastoja, jotka tuotiin osaksi projektia. Kirjastojen tuontia varten

tuli luoda erillinen header-tiedosto, jonka avulla kääntäjälle kerrotaan, missä projektissa käytettävät Objective-C-tiedostot sijaitsevat ja miten niitä käytetään [6].

Swiftiä on uudistettu monella tavalla verrattuna edeltäjäänsä. Esiprosessorin puuttumisen vuoksi esimerkiksi makroja ei voi enää käyttää. Syntaksia on nykyaikaistettu huomattavasti, ja siihen on haettu inspiraatioita moderneista kielistä (JavaScript, Ruby, C#). Esimerkiksi rivien loppuun lisättävä puolipisteen käyttö ei ole pakollista, vaikka se onkin mahdollista ja pääsyosoitin on vaihdettu pisteeksi entisen hakasulkujen sijaan. Näkyvyysalueen merkitseminen kaarisuluilla on Swftissä pakollista. Esimerkiksi ehtolausetta on seurattava sulut, joilla voidaan varmuudella määrittää lauseen näkyvyysalueen laajuus. [5]

3.2 Xcode-Ohjelmointiympäristö

Swift-kehitykselle luonnollisin kehitysympäristö on Applen Xcode [7]. Varsinkin kehitettäessä iOS-laittelle tai muille Applen tuotteille ei vaihtoehtoja juurikaan ole, sillä vaikka useita vaihtoehtoisia ohjelmointiympäristöjä on olemassa, ei niillä ole mahdollista ajaa koodia suoraan halutussa laitteessa. Valmiin ohjelman kääntämiseksi iOS-ohjelmaksi vaaditaan myös Xcodea, joten muilla ympäristöillä kehittäminen ei ole helppoa. Vaikka Swift onkin nykyään vapaata lähdekoodia [6], ei sama ajattelutapa koske itse kehitysympäristöä tai kehitystyötä. Xcode-ohjelmisto toimii ainoastaan Mac-tietokoneilla ja on ilmaiseksi ladattavissa. Ohjelmaa on mahdollista ajaa emulaattorissa ilmaiseksi, mutta jos ohjelman haluaa asentaa mobiililaitteelle, on ostettava kehittäjälisenssi [8].

Xcode tarjoaa perinteisen kehitysympäristön lisäksi käyttöliittymätyökalut ja iOS-emulaattorin [7]. Tarkoituksena Xcodessa on, että kehittäjä ei tarvitsisi sen ulkopuolisia työkaluja kehittäessään iOS-sovelluksia. Esimerkiksi työn pohjana toimineen sovelluksen tapauksessa tämä ei kuitenkaan toteudu, sillä osa ohjelmasta on toteutettu web-ohjelmana, joiden kehittämiseen Xcode ei tarjoa työkaluja. Ohjelmointiympäristö on monella tavalla samanlainen kuin monet kilpailijansa. Tarjolla on avainsanojen automaattinen täydennys, ja älykkäämpi korjaustoiminto, joka huomauttaa jo ohjelmakoodin kirjoitusvaiheessa mahdollisista virheistä tai huonoista käytännöistä [4]. Esimerkiksi tapauksessa, jossa luodaan muuttuja, jonka arvoa ei muuteta ohjelman aikana, Xcode ehdottaa käyttämään muuttujan sijaan vakiota. Korjaustoiminto huomauttaa myös, jos ohjelmoija käyttää vanhojen Swift-versioiden mukaista koodia ja tarjoaa korjausehdotuksia.

4. MOBIILITULOSTUKSEN TOTEUTUS

Työssä kuvatussa projektissa lisättiin jo olemassa olevaan iOS-ohjelman prototyyppiin mobiilitulostus. Alkuperäisessä ohjelmassa näytteenoton jälkeen hoitaja otti valmiin viivakooditarran, jonka hän tämän jälkeen skannasi mobiililaitteen kameralla ja kirjasi näytteen. Tästä haluttiin päästä eroon, sillä mobiililaitteen kameralla voi olla vaikeaa saada luettua viivakoodia, varsinkin huonossa valaistuksessa. Mobiilitulostuksella tuotiin viivakoodin lukemisen rinnalle vaihtoehtoinen tapa kirjata viivakoodin mukainen näyte järjestelmään. Mobiilitulostuksella toteutetussa vaihtoehdossa hoitaja tulostaa näytteen otettuaan viivakooditarran, jolloin tarkka näytteenottoaika saadaan tallennettua automaattisesti.

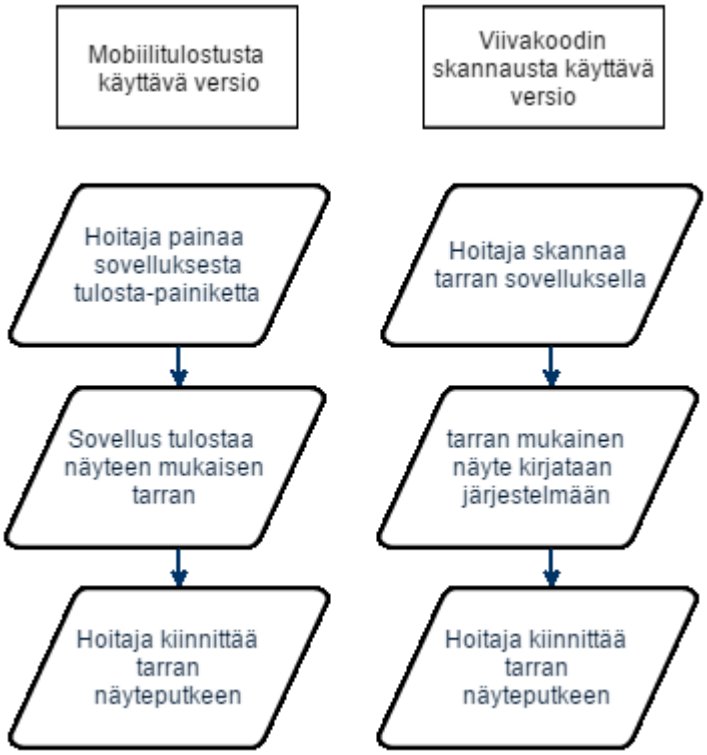
4.1 Ohjelman rakenne ja käytetyt sovelluskehykset

Ohjelma koostuu kahdesta osasta: pohjana toimii Swiftillä toteutettu natiiviohjelma, jonka sisällä ajetaan React-JavaScript-kirjastolla toteutettua web-ohjelmaa. Natiiviohjelma huolehtii ohjelman ja laitteen välisestä toiminnasta, kun taas web-ohjelmassa pitää sisällään käyttöliittymän ja näytteenkirjaussovelluksen. Web-ohjelmaa ajetaan WebKit-selainmoottorissa, joka noutaa web-ohjelman palvelimelta ja vastaa sen renderöinnistä [4].

Tulostimeen yhdistetään käyttäen Applen External Acessory -sovelluskehystä. Sovelluskehys on tarkoitettu mahdollistamaan ulkoisten laitteiden kanssa kommunikoiminen, liitäntätavasta riippumatta [6]. Tässä tapauksessa käytetään Bluetooth-rajapintaa. Mobiilitulostin ei tue Bluetooth Low Energy-verkkotekniikkaa, joten yhteyttä ei voi hallinnoida ohjelmasta käsin, vaan yhteyksiä on hallittava mobiililaitteen omilla ohjelmilla. Käytettävyyden kannalta ratkaisu on huonompi, sillä käyttäjä joutuu poistumaan ohjelmasta hallitakseen liitettyjä mobiilitulostimia. Ohjelman toteutuksen kannalta on kuitenkin helpompaa, että tulostimeen yhdistäminen suoritetaan muualla, jolloin ohjelman vastuulle jää enää mobiililaitteeseen yhdistetyistä laitteista halutun tulostimen valitseminen.

4.2 Ohjelman käyttö

Ohjelmaa käyttävä hoitaja saa sovellukselta tiedon potilaasta ja otettavasta näytteestä. Otettuaan näytteen hän tulostaa sovelluksen avulla viivakooditarran. Tarra kiinnitetään näyteputkeen ja näyte toimitetaan eteenpäin. Uusi toteutustapa nopeuttaa ja helpottaa näytteenottoprosessia, sillä aikaisemmin hoitajan on täytynyt tulostaa oikeanlainen tarra jo ennen potilaan luo siirtymistä, jolloin näytteenottoaikaa ei ole voitu lisätä tarraan automaattisesti.

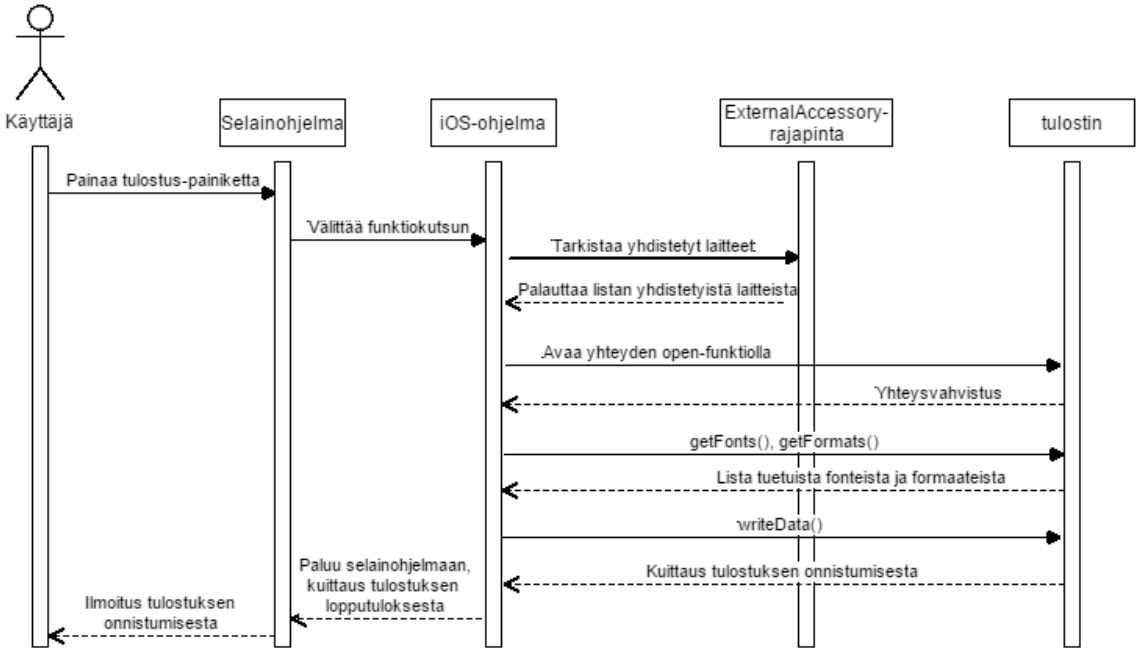


Kuva 4.1 Sovelluksen uuden ja vanhan version erot käyttäjän näkökulmasta

Nykyisellä toteutuksella tarraan saadaan automaattisesti oikea näytteenottoaika ja vältytään viivakoodin skannaamiselta. Skannaaminen on varsinkin vaihtelevien valaistusolosuhteiden vuoksi haastavaa, kun taas suoraan tulostaessa tätä ei tarvitse tehdä lainkaan. Skannausmahdollisuus jätetään kuitenkin vaihtoehdoksi, jotta näyte saadaan kirjattua, vaikka tulostimia ei olisi saatavilla tai ne olisivat epäkunnossa. Toteutusten erot on kuvattu kuvassa 4.1.

4.3 Ohjelman toiminta

Ohjelman varsinainen käyttöliittymä ja näytteiden käsittelyyn liittyvä toiminta on toteutettu web-ohjelmana, jota ajetaan selainmoottorissa, joten tulostuskäsky on välitettävä selainmoottorista natiiviohjelmalle. Koska selainmoottorissa ajettavalla web-ohjelmana ei ole näkyvyyttä selainmoottorin ulkopuoliseen Swift-koodiin, täytyy kaikki JavaScript-funktiot jotka välittävät tietoa varsinaisen iOS-sovelluksen puolelle sarjallistaa URL-osoitteiksi [9]. WkWebViewtä käytettäessä tämä tapahtuu automaattisesti, kunhan funktioita varten rekisteröidään käsittelijä. Samaan tapaan myös tulosta-painikkeen painamisesta aktivoituva funktio sarjallistetaan URL-osoitteeksi, joten kun käyttäjä painaa painiketta web-ohjelma kutsuu funktiolle määritettyä URL-osoitetta. Selainmoottori välittää kutsun iOS-sovellukselle joka vuorostaan purkaa URL-osoitteen ja kutsuu sille rekisteröityä Swift-funktiota.



Kuva 4.2 Mobiilitulostuksen eri osien välinen kommunikaatio

iOS-ohjelman tulostus-funktio vastaa yhteyden ottamisesta tulostimeen. Mobiililaitteella täytyy tässä vaiheessa olla jo Bluetooth-yhteys tulostimeen, jotta ohjelma voi etsiä tulostimen ExternalAccessoryn-rajapinnan kautta [6]. Jotta rajapinnan kautta saa haettua listan yhdistetyistä laitteista täytyy sille antaa tulostinta vastaava yhdistysmerkkijono. Rajapinnan antamasta listasta ohjelma hakee mobiilitulostimen, jos sellainen on yhdistetty. Jos tulostinta ei löydy keskeytetään funktion suorittaminen ja käyttäjää pyydetään muodostamaan yhteys mobiilitulostimeen ennen jatkamista. Sopivan tulostimen löytyessä avataan siihen yhteys. Yhteyden avaamiseksi on luotava tulostinvalmistajan sovelluskehityskirjaston rajapinnan mukainen yhteysolio, jonka open-jäsenfunktion avulla avataan yhteys tulostimeen [4]. Kun yhteys tulostimeen on muodostettu, ohjelma kysyy tulostimelta sen tukemat fontit getFonts-funktiolla ja tuetut formaatit getFormats-funktiolla. Näiden perusteella luodaan oikeanlaiset dokumentti- ja parametri-oliot, joilla muodostetaan halutunlainen tulostustyö. Tulostaminen tapahtuu yhteys-olion jäsenfunktiolla writeData, jolle määritetään tulostimelle kirjoitettava teksti [4]. Tulostustyön lähdettyä tulostimelle voidaan vapauttaa luotu dokumentti. Tämän jälkeen tulostus-funktion suoritus lopetetaan ja suoritus jatkuu web-ohjelman puolella. Viestien kulku ohjelman eri osien välillä on esitetty kuvassa 4.2.

5. ARVIOINTI JA MUIHIN TOTEUSTAPOHIN VERTAILU

Työssä kuvattua vastaavan mobiilisovelluksen voisi toteuttaa usealla eri tavalla. Valittu toteutustapa edustaa hybriditoteutusta, koska sovelluksen pohjana toimii natiiviohjelma, jonka sisällä ajetaan erillistä selainmoottoria, joka vastaa web-ohjelman toiminnasta. Valitun tavan lisäksi mobiilisovelluksia voidaan toteuttaa myös puhtaana natiiviohjelmana, kokonaan web-sovelluksena tai monialustasovelluskehysten avulla. Monialustakehyksellä toteutetussa ohjelmassa tuloksena on sovellus, joka on käännettävissä usealle eri alustalle.

5.1.1 Web-pohjainen mobiilisovellus

Web-tekniikoihin perustuvan mobiilikehityksen keskiössä on JavaScript, joka toisinkuin käännettävät kielet, tarvitsee erillisen sovelluskehyksen toimiakseen mobiililaitteessa [2]. Näistä yleisimpiä on WebKit, jota tämänkin työn projektissa käytetään. Webkitin sisäinen toteutus vaihtelee ja se voi pitää sisällään erilaisia moottoreita. Applen laitteissa käytetään yhtiön omaa JavaScriptCore-moottoria vastaamaan JavaScript-koodin ajamisesta laitteessa [2]. Web-pohjainen sovellus on yleensä mahdollista siirtää pienehköllä vaivalla laitteelta toiselle, sillä itse JavaScript-koodi ei muutu juurikaan. Heikkoutena web-pohjaisissa järjestelmissä on pienempi suorituskyky verrattuna natiivisovelluksiin, mutta tekniikka kehittyy jatkuvasti ja web-tekniikat ovat täysin varteenotettavia vaihtoehtoja mobiilikehityksessä, varsinkin jos ohjelma ei vaadi suorituskyvyltä ihmeitä.

Hyvänä esimerkkinä niin sanotut yhden sivun sovellukset (*one page app*), joissa ohjelman varsinainen toiminnallisuus on rajallista ja tähtää yleensä vain datan näyttämiseen, eikä niinkään sen muokkaamiseen. Web-tekniikat soveltuvat kuitenkin huonosti mobiilisovelluksiin joiden on tarkoitus käyttää paljon mobiililaitteen ominaisuuksia, kuten sensoreita tai likiyhteyksiä, sillä rajapinnat näiden käsittelemiseen web-kehyksen läpi ovat rajoittuneita.

5.1.2 Natiivisovellus

Natiivisovellus toteutetaan yleensä kohteena olevan mobiililaitteen mukaisilla kehitystyökaluilla (SDK), joita laitevalmistajat julkaisevat. Näin toteutetut ohjelmat ovat yleensä suorituskyvyltään parhaita mahdollisia, sillä ne voivat käyttää hyväksi kaikkia laitteen ominaisuuksia laitteen valmistajan haluamalla tavalla. Tästä johtuen ne ovat myös sidottuja yhdelle alustalle [10]. Työssä käsiteltävä mobiiliohjelma on osittain natiivisovellus, sillä se on kehitetty iOS-järjestelmien omalla kehityskielellä ja ohjelmaa ajetaan suoraan

laitteessa. Varsinaisesti kyse on kuitenkin hybridisovelluksesta, sillä sovelluksessa ajetaan erillistä web-ohjelmaa selainmoottorin avulla [8]. Natiivisovellukset ovat jokaisen alustan lähtökohtainen toteutustapa. Natiivitoteutuksella saa parhaiten käytettyä laitteiden kirjastoja ja siten sillä voidaan tehdä ohjelmia, jotka käyttävät laitteen kaikkia ominaisuuksia. Tästä syystä natiivisovellus on monissa tapauksissa ainoa vaihtoehto, jos sovelluksen on käytettävä laitteistoja, kuten kameraa tai erilaisia sensoreita. Myös suorituskyvyn ollessa tärkeä tekijä on paras vaihtoehto natiivisovellus.

5.1.3 Web-sovellus

Yhtenä vaihtoehtona voidaan pitää myös kokonaan selaimella käytettävää ohjelmaa. Toteutus on samankaltainen, kuin hybridisovelluksessa, mutta nyt sovellusta ei kapseloida erillisen natiivisovelluksen sisään, vaan ajetaan suoraan käytettävän mobiililaitteen selaimessa [10]. Tämä toteutustapa vapauttaa kehittäjän käytännössä kokonaan eri alustojen välisistä toteutuseroista, sillä sovelluksen toimintaan vaikuttaa pääasiassa selaimen ominaisuudet. Ohjelma on myös ajettavissa muidenkin laitteiden, kuten tietokoneiden selaimissa, muuttamatta sitä laisinkaan. Web-sovelluksilla voi olla kuitenkin vaikeaa, jopa mahdotonta, päästä käsiksi käytettävän laitteen ominaisuuksiin [10]. Tarkoitukseen on olemassa monille alustoille valmiita toteutuksia, mutta usein nämä joudutaan kuitenkin toteuttamaan itse, monimutkaistaen samalla projektia.

5.1.4 Monialustasovelluskehykset

Monialustasovelluskehyksillä pyritään tuottamaan koodia, joka on käännettävissä usealle eri alustalle. Kehitettäessä usealle alustalle samanaikaisesti vältytään saman ohjelman kirjoittamiselta uudelleen, jos halutaan siirtyä uudelle alustalle. Monialustasovelluskehyksiä on monenlaisia, joista saadaan yleensä lopputuloksena jokin aiemmin esittelyistä sovellustyypeistä. Yksinkertaisinta on tehdä hybridisovellus, sillä eri alustojen välillä muuttuu ainoastaan natiiviohjelma, jonka sisällä ajetaan jonkin sovelluskehyksen sisällä varsinaista ohjelmaa, joka on toteutettu jollain muulla kielellä [11]. Työssä esitellyssä sovelluksessa, kuten suurimmassa osassa monialustasovelluksiakin, ajetaan sovelluskehyksessä web-ohjelmaa. Monialustasovellusten hyvät ja huonot puolet vaihtelevat myös lopullisen ohjelman tyypin mukaan. Yhtenä etuna monialustasovelluskehyksissä on, että usealle alustalle kehitettäessä ei tarvitse hallita useaa eri kieltä, vaan kehitys tapahtuu samalla kiellellä joka alustalle. Natiivisovelluskehityksessä käytettävät kielet kuten Swift tai Objective-C, ovat monesti myös vähemmän käytettyjä, joten mahdollisuus kehittää ohjelmia tutummalla JavaScriptilla tai C++:lla houkuttaa kehittäjiä valitsemaan monialustaratkaisun. Osa monialustakehyksistä ei pysty täysin käyttämään kaikkia alustakohtaisia kirjastoja ja rajapintoja. Alustan omille kirjastoille on yleensä vähintään rajallinen tuki, mutta kolmannen osapuolen kirjastot ovat yleensä saatavilla vain alustan varsinaiselle kehityskielelle, joten niiden tuominen projektiin ei aina ole mahdollista. Hyvänä esimerkkinä monialustasovelluskehyksestä toimii Rhodes, jonka on kehittänyt Motorola

Solutions inc. yhteistyössä MIT:n kanssa. Kehityskielenä sovelluskehyksessä käytetään Rubya ja sillä voidaan kehittää mobiilisovelluksia iOS-, Android, BlackBerry, Windows Mobile ja Symbian-laitteille [12]. Rhodes käyttää MVC-mallia mobiilisovelluksen kehittämiseen, siten että Näkymät (*View*) on toteutettu HTML-, CSS ja JavaScript-kielillä, kun taas Rubylla toteutettu osa vastaa laitteen ja sovelluksen välisestä toiminnnasta.

Myös web-tekniikoiden ympärille on rakennettu monialustasovelluskehyksiä, joista käytetyin lienee Facebookin kehittämä React Native, joka perustuu React-websovelluskehykseen. Kehityskielenä React Native käyttää JavaScriptiä, mutta lopputuloksena on alustan mukainen natiivisovellus [13]. React Native on kuitenkin tällä hetkellä vasta versiossa 0.39, joten kehitettävää on vielä paljon. Sovelluskehys on kuitenkin jo laajassa käytössä, koska sitä käytettäessä ei tarvitse opetella eri alustojen omia kehityskieliä. Tämä helpottaa varsinkin web-kehittäjien siirtymistä mobiilikehityksen pariin, koska React ja sen kaltaiset JavaScript-pohjaiset web-sovelluskehykset kasvattavat suosiotaan jatkuvasti.

5.2 Sovelluksen arviointi ja vertailu muihin toteutusvaihtoehtoihin

Toteutettu toiminnallisuus vastaa lähtökohtaisesti kaikkia sille annettuja vaatimuksia. Ohjelma toimii ja on helppokäyttöinen. Tehokkuuden suhteenkin kaikki on kunnossa, sillä tulostuksessakaan ei ole mainittavaa viivettä. Nykyisen hybriditoteutuksen suurin ongelma on, että web-sovellusta ei ajeta paikallisesti, tai edes ladata käynnistyksen yhteydessä laitteeseen, vaan ohjelmaa vaatii jatkuvan internet-yhteyden toimiakseen. Luonnollisesti jonkinlainen verkkoyhteys on ohjelmalle pakollinen, sillä potilaiden ja näytteiden tiedot on haettava palvelimelta. Samoin otetun näytteen tiedot tulee lähettää palvelimelle viimeistään kierron päättyessä. Ero esimerkiksi puhtaaseen natiivitoteutukseen on huomattava, sillä nykyisessä toteutuksessa myös käyttöliittymä haetaan uudelleen aina siirryttäessä toiseen näkymään tai päivitettäessä tietoa. Ohjelman käyttö on siis käytännössä mahdotonta ilman internet-yhteyttä, vaikka tietoa ei lähetettäsikään laitteen ja palvelimen välillä. Natiivitoteutuksessa ainoastaan potilaiden ja näytteiden tietoja tarvitsisi liikuttaa palvelimelle ja takaisin, eikä sillä olisi merkitystä, milloin siirto tapahtuu. Myös tässä toteutuksessa voitaisiin yhteyden ollessa poikki tiedot tallentaa paikallisesti ja lähettää palvelimelle, kun yhteys on jälleen muodostettu. Täysin selainpohjainen toteutus voidaan sulkea heti pois, sillä tulostukseen ja skannaukseen vaaditaan omat kolmannen osapuolen kirjastonsa, joista ei ole toteutettu selaimessa käytettävää versiota [3]. Muilta osin sovellus olisi täysin toteuttavissa selainpohjaisena.

Natiivitoteutuksen etuna hybriditoteutukseen verrattuna on myös parempi suorituskyky, sillä erillisen JavaScript-pohjaisen ohjelman ajaminen sovelluskehyksen sisällä on hitaampaa kuin natiivikoodilla tuotettu ohjelma [10]. Kotinäytteenoton mobiilisovelluksen

tapauksessa tällä ei kuitenkaan ole käytännön merkitystä, sillä ohjelman pääasiallinen tarkoitus on vain välittää tiedot otetuista näytteistä ja näyttää näytetilaukset. Ainoa laitetta suuremmin käyttävä ominaisuus on juurikin mobiilitulostus, joka onkin toteutettu juuri natiivikoodilla. Natiivitoteutuksen huonona puolena on jatkokehityksen jäykkyys, sillä kun ohjelma on kertaalleen toteutettu jollekin alustalle natiivisti, ei sen siirtäminen muille alustoille ole kovinkaan yksinkertaista. Kaikilla mobiilialustoille on käytössään oma kielensä, kirjastonsa ja kehitysympäristönsä, jotka eivät keskustele keskenään käytännössä ollenkaan [10]. Nykyisellä toteutustavalla päästään huomattavasti helpommalla siirryttäessä uudelle alustalle, sillä suurin osa ohjelmasta pyörii muutenkin selainmoottorissa. Vastaavia selainmoottoreita löytyy kaikille varteenotettaville alustavaihtoehdoille, joten ainoastaan tulostuksen ja skannauksen kaltaiset mobiililaitteen laitteistoa käyttävät ominaisuudet on toteuttava uudelleen.

Monialustatoteutuksen ilmeisimpänä etuna nykyiseen toteutukseen nähden olisi mahdollisuus siirtyä uusille alustoille helposti ja vain pienin muutoksin. Tällä hetkellä käytössä olevan tulostimen kehityskirjastoista on olemassa C++-, Java-, Objective-C- ja C#-versiot, joten tulostin pitäisi olla täysin mahdollista saada toimimaan myös Androidilla ja esimerkiksi Windows Phonella. Kuten luvussa 5.1.5 todetaan monet monialustasovellukset ovat käytännössä hybridisovelluksia, joten suorituskyky olisi samaa luokkaa nykyisen toteutuksen kanssa, vaikka alustakohtaisia eroja olisi varmasti. Toisaalta käyttämällä alustalle tarkoitettua natiivikieltä ja kehitysympäristöä saadaan ohjelman natiivipuolesta enemmän irti, kuin jos käytetään monialustakehitysympäristöä, joka kääntää omalla kehityskielellään toteutetun ohjelman eri alustojen natiivikoodiksi [12]. Toisaalta nykyisenkin toteutuksen siirtäminen uusille alustoille suhteellisen helppoa. Tämä vähentää monialustatoteutuksen käytöstä saatavaa hyötyä. On myös olemassa monialustatoteutuksia, jotka eivät tuota hybridisovellusta, vaan tekevät kullekin laitteelle sopivaa natiivikoodia, kuten React Native [13]. Esimerkiksi React Nativea käytettäessä saadaan aikaiseksi yksi yhtenäinen pohjakoodi kaikille tuetuille alustoille, joiden alustakohtaisia eroja voidaan lisätä ja muokata erikseen. Myös selainmoottorin lisääminen on siten mahdollista. Selainmoottorin avulla toteutettu versio voisi olla hyvä välimuoto, jos sovellus päätetään muuttaa React Native-sovellukseksi. Työssä kuvatun projektin toteuttamiseen Monialustasovelluskehykset kelpaisivat myös, sillä ainakin esimerkkeinä käytetyt React Native ja Rhodes tukevat Objective-C-kirjastoja, joten työssä käytetyt tulostinkirjastot olisi mitä luultavammin mahdollista saada käyttöön myös niissä.

6. YHTEENVETO

Työssä suunniteltiin ja kehitettiin olemassa olevaan kotinäytteenoton mobiilisovellukseen mobiilitulostusominaisuus. Tämän lisäksi työssä arvoitiin mobiilisovelluksen kehityksen suunnitteluvalintoja, pääasiassa valittua toteutustapaa. Sovellus kehitettiin Applen Swift-ohjelmointikielellä iOS-laitteille ja Datamax Microflash 4te-tulostimelle. Sovelluksen kehitykseen käytettiin Xcode-Ohjelmointiympäristöä.

Lisätyn mobiilitulostuksen avulla näytteen ottava hoitaja voi helposti tulostaa viivakoodin aikaa vievän skannaamisen sijaan. Kierron valmisteluun kuluvassa ajassa säästetään myös, sillä hoitajan ei tarvitse enää tulostaa viivakoodeja valmiiksi ja liittää niitä oikeisiin näytetilauspyyntöihin. Sovelluksen käyttö on helppoa, eikä se ole muuttunut käyttäjän näkökulmasta suuresti. Skannausmahdollisuuskin on edelleen mukana, mahdollisia tulostimen toimintahäiriöitä silmällä pitäen. Tulostustoiminto toteutettiin mahdollisimman yksinkertaisesti nojaten vahvasti tulostinvalmistajan Objective-C:llä toteutettuihin esimerkkeihin. Lopputuloksena sovelluksen käyttö on helppoa ja sovelluksen toiminta on varmaa. Jatkokehityksessä tullaan toteuttamaan vähintään otettujen näytteiden tietojen paikallinen tallentaminen, jolloin mahdolliset verkkoyhteyden katkeamiset eivät häiritse ohjelman toimintaa.

Kotinäytteenoton mobiilisovellus on toteutettu hybridisovelluksena. Muita mahdollisuuksia olisivat natiivisovellus ja web-sovellus. Lisäksi sovellusta verrattiin monialustasovelluskehyksillä toteutettaviin ohjelmiin. Natiivisovellukseen nähden valittu toteutustapa on vaatinut vähemmän työtä, sillä alkuperäinen kotinäytteenottosovellus on toteutettu web-sovelluksena, joten sen käyttäminen selainmoottorin avulla on huomattavasti helpompaa kuin kokonaan uuden kehittäminen valitun alustan natiivikoodina. Natiivisovelluksen mukanaan tuomat hyödyt eivät myöskään ole kehitetyn sovelluksen kannalta merkittäviä, sillä laitteen suorituskyvyllä on todella vähän merkitystä. Web-sovellus ei lähtökohtaisesti olisi tullut lainkaan kysymykseen, sillä selaimella ei ole mahdollista käyttää tulostuksen ja skannauksen vaatimia kirjastoja [10]. Monialustatoteutukseessa suurin hyöty olisi mahdollisuus kääntää ohjelma usealle alustalle. Kehitetyn ohjelman näkökulmasta ainoastaan skannaus ja tulostus tulisi tässä tapauksessa toteuttaa uudelleen eri alustoille. Hyöty on kuitenkin hyvin pieni, sillä myös hybridisovellus on varsin helppo siirtää toisille alustoille, suurimman osan ohjelmasta ollessa alustariippumatonta web-koodia. Monialustasovelluskehityksessä on lisäksi vielä useita ongelmia ratkaistavanaan, ja vasta HTML5:n kehittyessä pidemmälle monialustakehitys saavuttaa täyden potentiaalinsa [10].

Voidaan todeta, että mobiilisovelluksen toteutustapa on toimiva ja lähtökohdat huomioon ottaen se on myös ollut helpoin toteuttaa. Koska alustavalintaan ei ohjelmaa kehitettäessä ole voinut vaikuttaa on Swiftin valinta ohjelmointikieleksi hyvin perusteltua sen nuoresta

iästä huolimatta. Varsinkin Applen aikomukset ajaa Swiftiä pääasialliseksi kehityskieleksi kaikille alustoilleen varmistaa, että tulevaisuudessa Swiftille tulee olemaan yhä enemmän ja enemmän käyttöä ja tukea myös kolmannen osapuolen laitteissa ja sovelluksissa [4]. Työtä tehdessä ei törmätty toimintahäiriöihin ja kielellä kehityt ohjelmat vaikuttavat vakailta ja turvallisilta. Kuitenkin suurimmat huolenaiheet kohdistuvat juurikin Swiftin nuorestä iästä johtuviin seikkoihin kuten tuen puutteeseen. Nykyisellään harva laitevalmistaja tarjoaa suoria kehitystyökaluja tai kirjastoja suoraan Swiftille, vaan ne on toteutettu Objective-C:llä. Objective-C:llä toteutettuja kirjastoja on kuitenkin mahdollista käyttää Swiftin kanssa, joten suurta ongelmaa ei tästä synny. Kehitysympäristö ja kieli itsessään ovat myös helppokäyttöisiä ja selkeitä.

LÄHTEET

[1] Petri Tahvanainen. ”Kotinäytteenoton varausjärjestelmän uudistaminen”, Tampereen teknillinen yliopisto, 2016. Saatavissa: http://URN.fi/URN:NBN:fi:tty-201603233747

[2] Andre Charland and Brian Leroux, "Mobile Application Development: Web vs. Native", *Commun. ACM, Volume 54 Issue 5*, 2011. pp. 49-53 Saatavissa: http://doi.acm.org/10.1145/1941487.1941504

[3] Datamax O’Neil iOS SDK version 2.2.7 Reference, 2014. Viitattu: 18.12.2016. Saatavissa: https://www.datamax-oneil.com/do/fr/en-us/home/support-downloads/downloads-drivers&p=ADB769FC-ADD9-E7A0-E433F70E0258156A#SM1

[4] Apple Developer Connection, 2016. Saatavissa: http://developer.apple.com/iphone/index.action. Viitattu 29.12.2016.

[5] Cristian G. García., Jordán P. Espada, B. C. P. G-Bustelo et al. “Swift vs. Objective-C: A New Programming Language”, University of Oviedo, Department of Computer Science, 2015 Saatavissa: http://www.ijimai.org/JOURNAL/sites/default/files/files/2015/05/ijimai20153_3_10_pdf_19818.pdf

[6] The Swift Programming Language (Swift 3), Apple Inc, 2016. Noudettu 18.9.2016. Saatavissa: https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/

[7] Wasserman, Anthony I. “Software Engineering Issues for Mobile Application Development”, ACM, 2010 pp. 397-400. Saatavissa: http://dl.acm.org/citation.cfm?id=1882443

[8] P. Smutný. "Mobile development tools and cross-platform solutions," Proceedings of the 13th International Carpathian Control Conference (ICCC), High Tatras, 2012, pp. 653-656. Saatavissa: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6228727&isnumber=6228605

[9] Thompson Matt. *WKWebView* (2014). Viitattu: 18.12.2016. Saatavissa: http://nshipster.com/wkwebkit/

[10] S. Amatya, A. Kurti, “Cross-Platform Mobile Development: Challenges and Opportunities”, in *Trajkovik V., Anastas M. (eds) ICT Innovations 2013*, *Advances in Intelligent Systems and Computing, vol 231.* Springer, Heidelberg, 2014 Saatavissa: http://link.springer.com/chapter/10.1007/978-3-319-01466-1_21

[11] Gustavo Hartmann, Geoff Stead, Asi DeGani. “Cross-platform mobile development”, Tribal, 2011 Saatavissa: https://wss.apan.org/jko/mole/Shared%20Documents/Cross-Platform%20Mobile%20Development.pdf

[12] M. Palmieri, I. Singh and A. Cicchetti, “Comparison of cross-platform mobile development tools”, 16th International Conference on Intelligence in Next Generation Networks, Berlin, 2012, pp. 179-186. Saatavissa: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6376023&isnumber=6376008

[13] React Native, Facebook Inc., 2016 https://facebook.github.io/react-native/docs/native-modules-ios.html Viitattu: 28.12.2016